Joshua Luis Mayores

Client/Server Reliable Chat Application: Final Report

With this application, I have implemented a server that can handle multiple clients connected to it, thus allowing the clients to communicate with each other through the server. To do this, I first created the server code.

Design Framework

```
import socket
import threading

HOST = socket.gethostname()
PORT = 9999
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((HOST, PORT))
clients = []
nicknames = []
```

To the left, we can see that I first have to create the socket that the server will be connected to, binding it to its specified host and port. There are two arrays that I then create: clients and nicknames. These will save the clients that connect to the server and their nicknames that they input.

This “broadcast_msg” function on the right takes whatever message a client has sent to the server and relays it to all the clients connected, using the “clients” array to do so.

```
# broadcast message to all clients
def broadcast_msg(message):
    for client in clients:
        client.send(message)
```

```
# handles clients
def handle_client(client, addr):
    connected = True
    while connected:
        msg = client.recv(1024)
        if msg:
            # exiting from server
            if msg.decode('ascii')[-7:] == ': .exit':
                connected = False
            else:
                broadcast_msg(msg)

    # removes client and nickname from server when exiting
    index = clients.index(client)
    nickname = nicknames[index]
    clients.remove(client)
    client.close()
    broadcast_msg(f"{nickname} has left the chatroom.".encode('ascii'))
    print(f"{nickname} has left the server.")
    nicknames.remove(nickname)
```

This “handle_client” function above simply handles all the clients’ messages and handles whenever they want to disconnect from the server. This function receives a message from a client, and if the message is the exit message, it leaves the while-loop. If not, it broadcasts the message. When it exits the while-loop, the server removes the client from its array, closes its socket, and broadcasts to all clients that the specified client has left the server.

```
def start_server():
    server.listen()
    while True:
        client, addr = server.accept()
        print("Connection has been established with a new client.")

        # gets nickname from client
        client.send("nickname".encode('ascii'))
        nickname = client.recv(1024).decode('ascii')
        nicknames.append(nickname)
        clients.append(client)
        print(f"The nickname of this client is {nickname}.")
        broadcast_msg(f"{nickname} has connected to the chatroom.".encode('ascii'))
        client.send("You are now connected to the chatroom.".encode('ascii'))

        # starts threading
        thread = threading.Thread(target=handle_client, args=(client, addr))
        thread.start()

print("Server is starting...")
start_server()
```

The final lines of code in the server code starts the server. It listens for any connection, and when a connection occurs, it accepts it and gets a nickname from the client. It saves it to the specified array and broadcasts to any clients connected that a new client is connected. Then a thread is started with the “handl_client” function to work together at the same time.

```
import socket
import threading
import sys

HOST = socket.gethostname()
PORT = 9999
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((HOST, PORT))

nickname = input("Choose a nickname: ")
```

Now for the code for the client. Similarly to the server code, we have to first create a socket that we then connect to with the specified host and port. It then asks the client to input a nickname.

```
# receives message from server
def client_receive():
    try:
        while True:
            message = client.recv(1024).decode('ascii')

            # if message is "nickname", sends client's nickname to server
            if message == "nickname":
                client.send(nickname.encode('ascii'))
            else:
                print(message)
    except:
        sys.exit()
```

This “client_receive” function handles any messages that the client receives from the server. It decodes the message into ascii first, and if the message coming from the server is “nickname”, it sends its nickname to the server. If not, it will just print out the message that a client has sent.

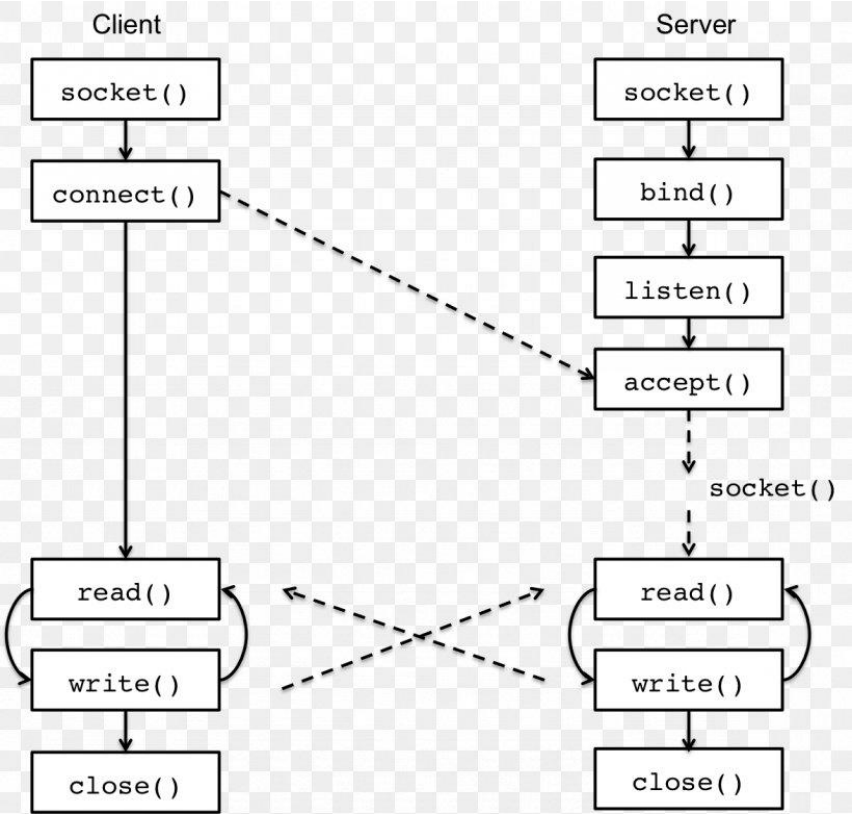
```
# sends message to server
def client_send():
    while True:
        message = f'{nickname}: {input("")}'
        client.send(message.encode('ascii'))

        # exiting
        if message == f'{nickname}: .exit':
            print("Disconnecting from server...")
            client.close()
            sys.exit()

# begins threading
receive_thread = threading.Thread(target=client_receive)
receive_thread.start()

send_thread = threading.Thread(target=client_send)
send_thread.start()
```

This “client_send” function obviously sends whatever message the client wants to send. It encodes it into ascii for the socket, and if the message is the exit message, it will print out to itself that it is disconnecting, closing its client and ending the code. Then I create the threads for the “client_receive” and “client_send” functions so they can run simultaneously.



Above is a visual flowchart to easily explain how the messages work and how the servers and clients function. On the server side, we create the socket, bind it to the specified host and port, and then listen to any connections. With the client, we create the socket and then connect it with the server socket. The server accepts the connection, then works to handle the client connections (named “write” in this diagram) and broadcasts any message sent (named “read” in this diagram). On the client side, it also works to send and receive (“read” and “write”) any message it creates or receives. Of course, the server and clients work together to encode and decode the messages with ascii to send them through the sockets.

Evaluation results

The installation process and running of my code is quite simple. After downloading my code, you may open a terminal that can run the server. You should then open 2 or more other terminals that will act as the client. For the terminal that will run the terminal, “cd” into the

specified folder where the code resides and then enter in “python server_program.py” to start the code. It should output this message:

```
C:\Users\joshl\PycharmProjects\COMP3825_Project>python server_pr
ogram.py
Server is starting...
```

Then to start the clients, do the same process where you “cd” into the folder and then enter in “python client_program.py” for each terminal that will act as the client. It will then ask for a nickname. Entering a nickname will confirm that you are connected to the server. On the server side, it will say that a new client has connected alongside the nickname you have chosen.

Client side:

```
C:\Users\joshl\PycharmProjects\COMP3825_Project>python client_pr
ogram.py
Choose a nickname: Joshua
Joshua has connected to the chatroom.
You are now connected to the chatroom.
```

Server side:

```
Connection has been established with a new client.
The nickname of this client is Joshua.
```

Repeat the same process for all other clients you wish to create and it will repeat the same outputs as shown above.

```
(c) Microsoft Corporation. All rights reserved.

C:\Users\joshl>cd PycharmProjects\COMP3825_Project

C:\Users\joshl\PycharmProjects\COMP3825_Project>python client_pr
ogram.py
Choose a nickname: Joshua
Joshua has connected to the chatroom.
You are now connected to the chatroom.
John has connected to the chatroom.
John: Hello there
What's up?
Joshua: What's up?
John: Nothing much
```

```
(c) Microsoft Corporation. All rights reserved.

C:\Users\joshl>cd PycharmProjects\COMP3825_Project

C:\Users\joshl\PycharmProjects\COMP3825_Project>python client_pr
ogram.py
Choose a nickname: John
John has connected to the chatroom.
You are now connected to the chatroom.
Hello there
John: Hello there
Joshua: What's up?
Nothing much
John: Nothing much
```

Above is a sample showing the use of the program. We have 2 clients, one named Joshua and another named John. We can see that typing in your message will send to the other client and yourself to ensure that your message was sent.

```
C:\Users\joshl\PycharmProjects\COMP3825_Project>python client_pr
ogram.py
Choose a nickname: Joshua
Joshua has connected to the chatroom.
You are now connected to the chatroom.
John has connected to the chatroom.
John: Hello there
What's up?
Joshua: What's up?
John: Nothing much
John has left the chatroom.
```

```
C:\Users\joshl\PycharmProjects\COMP3825_Project>python client_pr
ogram.py
Choose a nickname: John
John has connected to the chatroom.
You are now connected to the chatroom.
Hello there
John: Hello there
Joshua: What's up?
Nothing much
John: Nothing much
.exit
Disconnecting from server...
```

When you enter in the exit message, it will show that you are disconnecting from the server on your side alongside the other clients. We can see that it says “John has left the chatroom.”

```
John has left the server.
```

On the server side, it shows that John has left the server.